

Chapter 11

Object and Object- Relational Databases



Addison-Wesley
is an imprint of

PEARSON

Chapter 11 Outline

- Overview of Object Database Concepts
- Object-Relational Features:
Object Database Extensions to SQL
- The ODMG Object Model and the Object Definition Language ODL
- Object Database Conceptual Design
- The Object Query Language OQL
- Overview of the C++ Language Binding in the ODMG Standard

Object and Object-Relational Databases

- **Object databases (ODB)**
 - **Object data management systems (ODMS)**
 - Meet some of the needs of more complex applications
 - Specify:
 - Structure of complex objects
 - Operations that can be applied to these objects

Overview of Object Database Concepts

- Introduction to object-oriented concepts and features
 - Origins in OO programming languages
 - Object has two components:
 - State (value) and behavior (operations)
 - Instance variables
 - Hold values that define internal state of object
 - Operation is defined in two parts:
 - Signature or interface and implementation

Overview of Object Database Concepts (cont'd.)

- Inheritance
 - Permits specification of new types or classes that inherit much of their structure and/or operations from previously defined types or classes
- Operator overloading
 - Operation's ability to be applied to different types of objects
 - Operation name may refer to several distinct implementations

Object Identity, and Objects versus Literals

- Unique identity
 - Implemented via a unique, system-generated object identifier (OID)
 - **Immutable**
- Most OO database systems allow for the representation of both objects and literals (or values)

Complex Type Structures for Objects and Literals

- Structure of arbitrary complexity
 - Contain all necessary information that describes object or literal
- Nesting **type constructors**
 - Construct complex type from other types
- Most basic constructors:
 - **Atom**
 - **Struct (or tuple)**
 - **Collection**

Complex Type Structures for Objects and Literals (cont'd.)

- Collection types:
 - Set
 - Bag
 - List
 - Array
 - Dictionary
- **Object definition language (ODL)**
 - Used to define object types for a particular database application

Figure 11.1

Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

define type EMPLOYEE

```
tuple (  Fname:    string;
         Minit:    char;
         Lname:    string;
         Ssn:      string;
         Birth_date: DATE;
         Address:  string;
         Sex:      char;
         Salary:   float;
         Supervisor: EMPLOYEE;
         Dept:     DEPARTMENT;
```

define type DATE

```
tuple (  Year:    integer;
         Month:   integer;
         Day:     integer; );
```

define type DEPARTMENT

```
tuple (  Dname:    string;
         Dnumber: integer;
         Mgr:      tuple (  Manager:  EMPLOYEE;
                           Start_date: DATE; );
         Locations: set(string);
         Employees: set(EMPLOYEE);
         Projects:  set(PROJECT); );
```

Encapsulation of Operations and Persistence of Objects

■ Encapsulation

- Related to abstract data types and information hiding in programming languages
- Define **behavior** of a type of object based on operations that can be externally applied
- External users only aware of interface of the operations
- Divide structure of object into visible and hidden attributes

Encapsulation of Operations

- **Object constructor**
 - Used to create a new object
- **Destructor operation**
 - Used to destroy (delete) an object
- **Modifier operations**
 - Modify the states (values) of various attributes of an object
- **Retrieve** information about the object
- Dot notation used to apply operations to object

Persistence of Objects

- **Transient objects**

- Exist in executing program
- Disappear once program terminates

- **Persistent objects**

- Stored in database and persist after program termination
- **Naming mechanism**
- **Reachability**

Type Hierarchies and Inheritance

- Inheritance
 - Definition of new types based on other predefined types
 - Leads to **type** (or **class**) hierarchy
- Type: **type name** and list of visible (public) **functions**
 - Format:
 - `TYPE_NAME: function, function, ..., function`

Type Hierarchies and Inheritance (cont'd.)

■ Subtype

- Useful when creating a new type that is similar but not identical to an already defined type
- Example:
 - `EMPLOYEE` subtype-of `PERSON`: `Salary`, `Hire_date`, `Seniority`
 - `STUDENT` subtype-of `PERSON`: `Major`, `Gpa`

Type Hierarchies and Inheritance (cont'd.)

■ **Extent**

- Store collection of persistent objects for each type or subtype
- Extents are subsets of the extent of class OBJECT

■ **Persistent collection**

- Stored permanently in the database

■ **Transient collection**

- Exists temporarily during the execution of a program

Other Object-Oriented Concepts

- **Polymorphism** of operations
 - Also known as **operator overloading**
 - Allows same operator name or symbol to be bound to two or more different implementations
 - Depending on type of objects to which operator is applied
- **Multiple inheritance**
 - Subtype inherits functions (attributes and methods) of more than one supertype

Other Object-Oriented Concepts (cont'd.)

- **Selective inheritance**
 - Subtype inherits only some of the functions of a supertype

Summary of Object Database Concepts

- Object identity
- Type constructor
- Encapsulation of operations
- Programming language compatibility
- Type hierarchies and inheritance
- Extents
- Polymorphism and operator overloading

Object-Relational Features: Object Database Extensions to SQL

- **Type constructors**
 - Specify complex objects
- Mechanism for specifying **object identity**
- **Encapsulation of operations**
 - Provided through user-defined types (UDTs)
- **Inheritance mechanisms**
 - Provided using keyword UNDER

User-Defined Types and Complex Structures for Objects

- **UDT syntax:**

- `CREATE TYPE TYPE_NAME AS
(<component declarations>);`

- **ROW TYPE**

- Directly create a structured attribute using the keyword **ROW**

User-Defined Types and Complex Structures for Objects (cont'd.)

- Array type
 - Reference elements using []
- **CARDINALITY** function
 - Return the current number of elements in an array

Object Identifiers Using Reference Types

- **Reference type**

- Create unique system-generated object identifiers
- Examples:
 - REF IS SYSTEM GENERATED
 - REF IS <OID_ATTRIBUTE>
<VALUE_GENERATION_METHOD> ;

Creating Tables Based on the UDTs

- **INSTANTIABLE**

- Specify that UDT is instantiable
- Causes one or more tables to be created

Encapsulation of Operations

- User-defined type
 - Specify methods (or operations) in addition to the attributes
 - Format:

```
CREATE TYPE <TYPE-NAME> (  
  <LIST OF COMPONENT ATTRIBUTES AND THEIR TYPES>  
  <DECLARATION OF FUNCTIONS (METHODS)>  
) ;
```


Encapsulation of Operations (cont'd.)

- Constructor function **TYPE_T ()**
 - Returns a new object of that type
 - Format

```
DECLARE EXTERNAL <FUNCTION_NAME>  
<SIGNATURE>  
LANGUAGE <LANGUAGE_NAME>;
```

Specifying Inheritance and Overloading of Functions

- Inheritance rules:
 - All attributes inherited
 - Order of supertypes in UNDER clause determines inheritance hierarchy
 - Instance of a subtype can be used in every context in which a supertype instance used
 - Subtype can redefine any function defined in supertype

Specifying Inheritance and Overloading of Functions (cont'd.)

- When a function is called, best match selected based on types of all arguments
- For dynamic linking, runtime types of parameters is considered

Specifying Relationships via Reference

- Component attribute of one tuple may be a **reference** to a tuple of another table
 - Specified using keyword **REF**
- Keyword **SCOPE**
 - Specify name of table whose tuples referenced
- **Dot notation**
 - Build path expressions
- **→**
 - Used for dereferencing

The ODMG Object Model and the Object Definition Language ODL

- ODMG object model
 - Data model for **object definition language (ODL)** and **object query language (OQL)**
- Objects and Literals
 - Basic building blocks of the object model
- Object has five aspects:
 - **Identifier, name, lifetime, structure, and creation**
- **Literal**
 - Value that does not have an object identifier

The ODMG Object Model and the ODL (cont'd.)

- **Behavior** refers to operations
- **State** refers to properties
- **Interface**
 - Specifies only behavior of an object type
 - Typically **noninstantiable**
- **Class**
 - Specifies both state (attributes) and behavior (operations) of an object type
 - **Instantiable**

Inheritance in the Object Model of ODMG

- **Behavior inheritance**
 - Also known as IS-A or interface inheritance
 - Specified by the colon (:) notation
- **EXTENDS inheritance**
 - Specified by keyword **extends**
 - Inherit both state and behavior strictly among classes
 - Multiple inheritance via extends not permitted

Built-in Interfaces and Classes in the Object Model

- **Collection objects**
 - Inherit the basic Collection interface
- `I = O.create_iterator()`
 - Creates an iterator object for the collection
- Collection objects further specialized into:
 - `set`, `list`, `bag`, `array`, and `dictionary`

Built-in Interfaces and Classes in the Object Model (cont'd.)

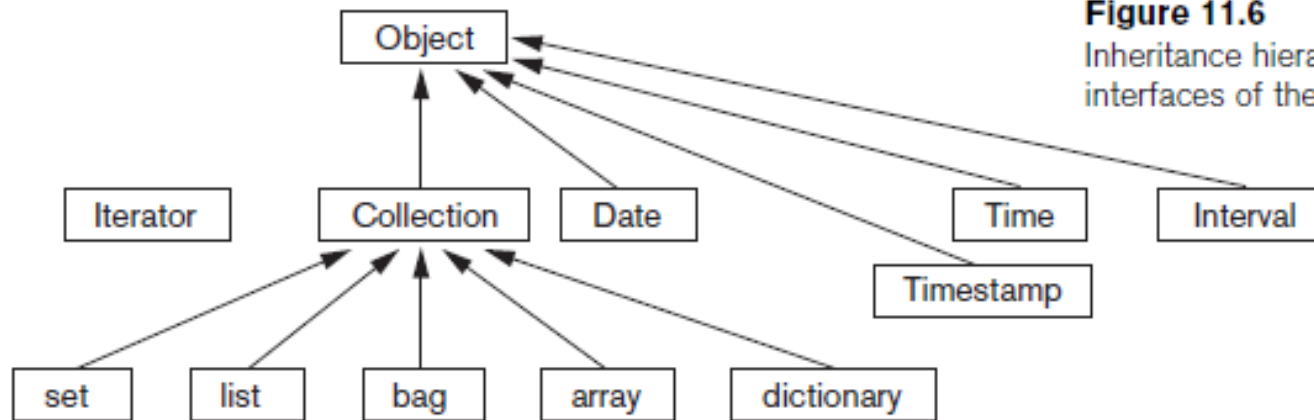


Figure 11.6

Inheritance hierarchy for the built-in interfaces of the object model.

Atomic (User-Defined) Objects

- Specified using keyword **class** in ODL
- **Attribute**
 - Property; describes some aspect of an object
- **Relationship**
 - Two objects in the database are related
 - Keyword **inverse**
 - Single conceptual relationship in inverse directions
- **Operation signature:**
 - Operation name, argument types, return value

Extents, Keys, and Factory Objects

- **Extent**

- Contains all persistent objects of class

- **Key**

- One or more properties whose values are unique for each object in extent

- **Factory object**

- Used to generate or create individual objects via its operations

The Object Definition Language ODL

- Support semantic constructs of ODMG object model
- Independent of any particular programming language

Figure 11.10

Possible ODL schema for the UNIVERSITY database in Figure 11.8(b).

```

class PERSON
(
  extent    PERSONS
  key       Ssn )
{
  attribute struct Pname {   string  Fname,
                             string  Mname,
                             string  Lname }   Name;

  attribute string          Ssn;
  attribute date            Birth_date;
  attribute enum Gender{M, F} Sex;
  attribute struct Address { short  No,
                             string Street,
                             short  Apt_no,
                             string City,
                             string State,
                             short  Zip }   Address;

  short  Age(); };
class FACULTY extends PERSON
(
  extent    FACULTY )
{
  attribute string      Rank;
  attribute float       Salary;
  attribute string      Office;
  attribute string      Phone;
  relationship DEPARTMENT Works_in inverse DEPARTMENT::Has faculty;
  relationship set<GRAD_STUDENT> Advises inverse GRAD_STUDENT::Advisor;
  relationship set<GRAD_STUDENT> On_committee_of inverse GRAD_STUDENT::Committee;
  void        give_raise(in float raise);
  void        promote(in string new rank); };
class GRADE
(
  extent    GRADES )
{
  attribute enum GradeValues{A,B,C,D,F,I, P} Grade;
  relationship SECTION Section inverse SECTION::Students;
  relationship STUDENT Student inverse STUDENT::Completed_sections; };

```

Object Database Conceptual Design

- Differences between conceptual design of ODB and RDB, handling of:
 - Relationships
 - Inheritance
- Philosophical difference between relational model and object model of data
 - In terms of behavioral specification

Mapping an EER Schema to an ODB Schema

- Create ODL class for each EER entity type
- Add relationship properties for each binary relationship
- Include appropriate operations for each class
- ODL class that corresponds to a subclass in the EER schema
 - Inherits type and methods of its superclass in ODL schema

Mapping an EER Schema to an ODB Schema (cont'd.)

- Weak entity types
 - Mapped same as regular entity types
- Categories (union types)
 - Difficult to map to ODL
- An n -ary relationship with degree $n > 2$
 - Map into a separate class, with appropriate references to each participating class

The Object Query Language OQL

- Query language proposed for ODMG object model
- Simple OQL queries, database entry points, and iterator variables
 - Syntax: select ... from ... where ... structure
 - Entry point: named persistent object
 - Iterator variable: define whenever a collection is referenced in an OQL query

Query Results and Path Expressions

- Result of a query
 - Any type that can be expressed in ODMG object model
- OQL orthogonal with respect to specifying path expressions
 - Attributes, relationships, and operation names (methods) can be used interchangeably within the path expressions

Other Features of OQL

- **Named query**
 - Specify identifier of named query
- OQL query will return collection as its result
 - If user requires that a query only return a single element use **element** operator
- Aggregate operators
- Membership and quantification over a collection

Other Features of OQL (cont'd.)

- Special operations for ordered collections
- **Group by** clause in OQL
 - Similar to the corresponding clause in SQL
 - Provides explicit reference to the collection of objects within each group or **partition**
- **Having clause**
 - Used to filter partitioned sets

Overview of the C++ Language Binding in the ODMG Standard

- Specifies how ODL constructs are mapped to C++ constructs
- Uses prefix `d_` for class declarations that deal with database concepts
- Template classes
 - Specified in library binding
 - Overloads operation `new` so that it can be used to create either persistent or transient objects

Summary

- Overview of concepts utilized in object databases
 - Object identity and identifiers; encapsulation of operations; inheritance; complex structure of objects through nesting of type constructors; and how objects are made persistent
- Description of the ODMG object model and object query language (OQL)
- Overview of the C++ language binding